Laporan Tugas Kecil 1

IF2211 Strategi Algoritma

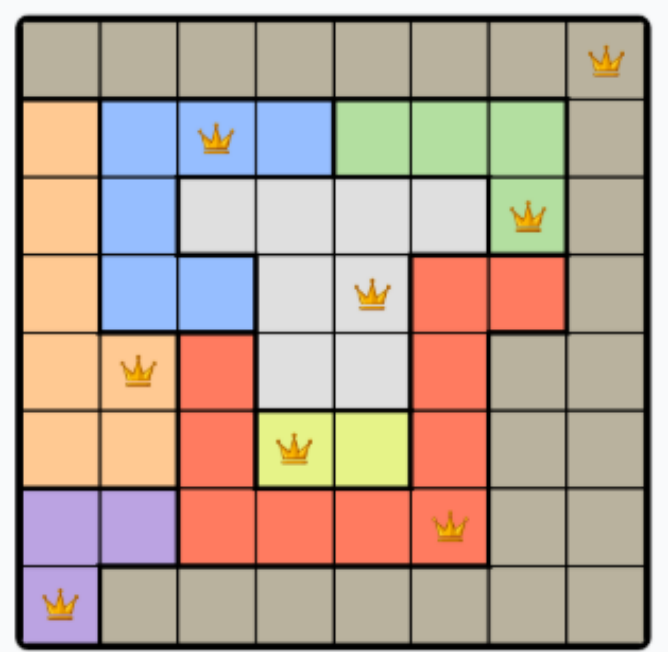
Penyelesaian Permainan Queens Linkedin

Muhammad Azzam Robbani - 18223025
18223025@std.stei.itb.ac.id

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2026

Daftar Isi

Deskripsi Permasalahan



Permainan **Queens** dimulai dengan sebuah papan persegi kosong dengan berbagai daerah warna. Pemain diharuskan untuk menempatkan queen sehingga terdapat hanya satu queen pada tiap baris, kolom, dan daerah warna. Selain itu, satu queen tidak dapat ditempatkan bersebelahan dengan queen lainnya, termasuk secara diagonal.

Tugas ini mengharuskan pencarian solusi menggunakan algoritma brute force, yaitu mencoba semua kemungkinan penempatan queen hingga solusi yang memenuhi semua syarat ditemukan.

Output yang diharapkan adalah:

1. Menampilkan hasil akhir dari papan yang sudah terisi oleh queens dengan aturan yang benar.
2. Menampilkan banyak konfigurasi atau iterasi yang ditinjau oleh algoritma.
3. Menampilkan waktu eksekusi program dalam milisecond (cukup waktu pencarian dengan algoritma, tidak termasuk membaca dan menulis file).
4. Live Update: program harus dapat memvisualisasikan proses brute force yang dilakukan.

Pendekatan Solusi

Algoritma brute force yang saya gunakan disini berbasis permutasi. Papan berukuran NxN direpresentasikan dengan sebuah array permutasi, di mana indeks menyatakan baris dan nilai di dalamnya menyatakan kolom posisi queen, sehingga secara otomatis setiap baris dan kolom hanya memiliki satu queen. Ketika program diberi input txt, program akan menghasilkan seluruh kemungkinan permutasi kolom. Untuk setiap permutasi, dilakukan pengecekan constraint yaitu tidak boleh ada dua queen dalam warna yang sama, tidak boleh ada dua queen dalam satu baris atau kolom, dan tidak boleh ada dua queen pada diagonal berdekatan. Jika suatu permutasi memenuhi seluruh constraint, maka solusi ditemukan.

Secara singkat, berikut adalah urutan penyelesaiannya:

1. Inisiasi brute force (membuat permutasi awal dan memulai pencarian)
2. Cek validitas (tidak boleh ada queen dalam satu warna dan tidak boleh ada queen yang bertetangga secara diagonal)
3. Jika valid, solusi ditemukan
4. Jika tidak valid, maka dibuat permutasi baru untuk dicek kembali

Implementasi dalam Kode

1. Menginisiasi Brute Force

Kode:

```
def start(self):
        if not self.board:
            messagebox.showerror("Error", "Load file terlebih
dahulu!")
            return

        self.running =True
        self.count = 0
        self.starttime = time.time()
        self.perm = list(range(self.n))
        self.answer = None
        self.solve()
```

Penjelasan:

Proses brute force dimulai dengan membuat permutasi awal berupa urutan kolom [0,1,2,...,n-1], lalu proses pencarian solusi dimulai dengan memanggil fungsi solve().

2. Mengecek Kevalidan Suatu Permutasi

Kode:

```
def solve(self):
        if not self.running:
            return

        for _ in range(self.update_interval):
            self.count += 1

            if areachecker(self.perm, self.board) and
diagonalchecker(self.perm):
                self.running = False
                self.answer = self.perm[:]
                end = time.time()
                self.draw(self.answer)
                self.stats_label.config(
                    text=f"Solution ~ {self.count}
permutations ~ {(end-self.starttime)*1000:.2f} ms"
                )
                return

            if not nextperm(self.perm):
                self.running = False
                end = time.time()
                self.stats_label.config(
                    text=f"No Solution ~ {self.count}
permutations ~ {(end-self.starttime)*1000:.2f} ms"
                )
                return

        self.draw(self.perm)
        self.stats_label.config(text=f"Checked: {self.count}")
        self.root.after(1, self.solve)
```

Penjelasan:

Setiap permutasi yang dihasilkan langsung diperiksa validitasnya dengan memanggil fungsi areachecker dan diagonalchecker. Jika keduanya bernilai True, maka solusi ditemukan dan proses dihentikan.

3. Memastikan Tidak Ada Dua Queen dalam Satu Warna

Kode:

```
def areachecker(perm, board):
    block= set()
    for r in range(len(perm)):
        area = board[r][perm[r]]
        if area in block:
            return False
        block.add(area)

    return True
```

Penjelasan:

Program memastikan setiap queen berada pada warna yang berbeda dengan menyimpan area yang sudah ditempati dalam sebuah set. Jika ditemukan dua queen berada pada warna yang sama, maka permutasi dianggap tidak valid.

4. Memastikan Tidak Ada Dua Queen pada Diagonal Berdekatan

Kode:

```
def diagonalchecker(perm):
    for i in range(len(perm) - 1):
        if abs(perm[i] - perm[i+1]) == 1:
            return False

    return True
```

Penjelasan:

Program memeriksa apakah terdapat dua queen pada baris yang bersebelahan dengan selisih kolom sebesar 1. Jika selisih kolom antar baris bertetangga sama dengan 1, maka kedua queen berada pada diagonal berdekatan dan permutasi dianggap tidak valid.

5. Menghasilkan Permutasi Baru Bila Solusi Belum Ditemukan

Kode:

```
def nextperm(arr):
    n = len(arr)
    i = n -2

    while i >= 0 and arr[i] >= arr[i+1]:
        i-= 1

    if i <0:
        return False

    j= n - 1

    while arr[j] <= arr[i]:
        j -= 1
    arr[i], arr[j] = arr[j], arr[i]
    arr[i+ 1:] = reversed(arr[i+1:])

    return True
```

Penjelasan:

Algoritma ini menghasilkan satu permutasi berikutnya setiap kali dipanggil hingga seluruh kemungkinan n! permutasi habis diperiksa.

Analisis Kompleksitas

Algoritma ini menggunakan pendekatan brute force dengan menghasilkan seluruh permutasi kolom sebanyak n!. Untuk setiap permutasi, dilakukan pengecekan

validitas area dan diagonal yang masing-masing memiliki kompleksitas O(n). Selain itu, proses pembangkitan permutasi berikutnya juga membutuhkan O(n). Maka, kompleksitas waktu keseluruhan algoritma adalah $O(n! \cdot n)$, sedangkan kompleksitas ruang adalah $O(n^2)$.

Bonus

1. GUI

Kode:

```
class GUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Queen Solver")
        self.board = []
        self.n = 0
        self.perm = []
        self.answer = None
        self.count = 0
        self.running = False
        self.starttime = 0
        self.colors = {}
        self.filename = None
        self.update_interval = 2000
        top = tk.Frame(root)
        top.pack(pady=5)

        tk.Button(top, text="Load", command=self.load,
width=10).grid(row=0, column=0, padx=5)
        tk.Button(top, text="Solve", command=self.start,
width=10).grid(row=0, column=1, padx=5)
        tk.Button(top, text="Save Image",
command=self.saveimg, width=12).grid(row=0, column=2, padx=5)
        tk.Button(top, text="Save Text", command=self.savetxt,
width=12).grid(row=0, column=3, padx=5)
        self.info_label = tk.Label(root, text="")
        self.info_label.pack()
        self.stats_label = tk.Label(root, text="")
        self.stats_label.pack()
        self.canvas = tk.Canvas(root)
        self.canvas.pack(pady=5)

    def colorgenerator(self):
        differentcolors = set(sum(self.board, []))
```

```
        self.colors = {}
        for area in differentcolors:
            r = random.randint(180,255)
            g = random.randint(180, 255)
            b = random.randint(180, 255)
            self.colors[area] = f'#{r:02x}{g:02x}{b:02x}'

    def load(self):
        filepath =
filedialog.askopenfilename(filetypes=[("Text files",
"*.txt")])
        if not filepath:
            return

        self.filename =os.path.basename(filepath)
        with open(filepath, 'r') as f:
            self.board = [list(line.strip()) for line in f if
line.strip()]

        self.n = len(self.board)

        for row in self.board:
            if len(row) !=self.n:
                messagebox.showerror("Error", "Board harus
NxN")
                return

        self.perm = list(range(self.n))
        self.answer = None
        self.colorgenerator()
        size= min(600, 60 * self.n)
        self.canvas.config(width=size, height=size)
        self.draw()
        self.info_label.config(text=f"Board {self.n}x{self.n}
loaded")

    def draw(self, perm=None):
        self.canvas.delete("all")
        if self.n == 0:
            return

        canvassize = int(self.canvas["width"])
        cell = canvassize // self.n

        for r in range(self.n):
            for c in range(self.n):
                x1 = c* cell
                y1 = r*cell
                x2 = x1+ cell
                y2 = y1+ cell
                area = self.board[r][c]
```

```
                color = self.colors.get(area, "white")
                self.canvas.create_rectangle(
                    x1, y1, x2, y2,
                    fill=color,
                    outline="black"
                )

                if perm and perm[r] == c:
                    self.canvas.create_text(
                        x1 + cell/2,
                        y1 +cell/2,
                        text="#",
                        font=("Arial", int(cell/2), "bold"),
                        fill="black"
                    )
                else:
                    self.canvas.create_text(
                        x1+cell/2,
                        y1+cell/2,
                        text=area,
                        font=("Arial", int(cell/3))
                    )

    def start(self):
        if not self.board:
            messagebox.showerror("Error", "Load file terlebih
dahulu!")
            return

        self.running =True
        self.count = 0
        self.starttime = time.time()
        self.perm = list(range(self.n))
        self.answer = None
        self.solve()

    def solve(self):
        if not self.running:
            return

        for _ in range(self.update_interval):
            self.count += 1

            if areachecker(self.perm, self.board) and
diagonalchecker(self.perm):
                self.running = False
                self.answer = self.perm[:]
                end = time.time()
                self.draw(self.answer)
                self.stats_label.config(
                    text=f"Solution ~ {self.count}
```

```
permutations ~ {(end-self.starttime)*1000:.2f} ms"
                )
                return

            if not nextperm(self.perm):
                self.running = False
                end = time.time()
                self.stats_label.config(
                    text=f"No Solution ~ {self.count}
permutations ~ {(end-self.starttime)*1000:.2f} ms"
                )
                return

        self.draw(self.perm)
        self.stats_label.config(text=f"Checked: {self.count}")
        self.root.after(1, self.solve)

    def savetxt(self):
        if self.answer is None:
            messagebox.showerror("Error", "Belum ada solusi!")
            return

        dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
        savefolder = os.path.join(dir, "test")
        basename = self.filename.replace(".txt", "")
        savepath = os.path.join(savefolder,
f"{basename}_solution.txt")

        with open(savepath, "w") as f:
            for r in range(self.n):
                row = ""
                for c in range(self.n):
                    if self.answer[r] == c:
                        row += "#"
                    else:
                        row += self.board[r][c]
                f.write(row+"\n")

        messagebox.showinfo("Saved", f"Saved to:\n{savepath}")

    def saveimg(self):
        if self.answer is None:
            messagebox.showerror("Error", "Belum ada solusi!")
            return

        dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
        savefolder = os.path.join(dir, "test")
        basename = self.filename.replace(".txt", "")
        savepath = os.path.join(savefolder,
```

```
f"{basename}_solution.png")
        size = 80
        imgsize = self.n*size
        image = Image.new("RGB", (imgsize, imgsize), "white")
        draw = ImageDraw.Draw(image)

        try:
            font = ImageFont.truetype("arial.ttf",
int(size/2))
        except:
            font = ImageFont.load_default()

        for r in range(self.n):
            for c in range(self.n):
                x1 = c*size
                y1 = r*size
                x2 = x1+size
                y2 = y1+size

                area = self.board[r][c]
                color = self.colors.get(area, "#ffffff")

                draw.rectangle([x1, y1, x2, y2], fill=color,
outline="black")

                if self.answer[r] == c:
                    text = "#"
                else:
                    text = area

                bbox = draw.textbbox((0, 0), text, font=font)
                w= bbox[2]-bbox[0]
                h = bbox[3]-bbox[1]

                draw.text(
                    (x1+(size-w)/2, y1+(size-h)/2),
                    text,
                    fill="black",
                    font=font
                )

        image.save(savepath)
        messagebox.showinfo("Saved", f"Saved to:\n{savepath}")
```

Penjelasan:

GUI dibangun menggunakan library Tkinter sebagai framework GUI utama.

Pada saat objek GUI diinisialisasi, program membuat tampilan utama dengan beberapa button, yaitu tombol Load, Solve, Save Image, dan Save Text. Selain itu, terdapat label informasi untuk menampilkan status papan dan statistik pencarian, serta sebuah area kosong yang digunakan untuk menampilkan visualisasi papan permainan.

Ketika pengguna menekan tombol Load, program membaca file teks berisi representasi papan, menyimpannya dalam bentuk matriks dua dimensi, lalu memvalidasi bahwa papan berbentuk NxN. Setelah itu, sistem menghasilkan warna acak untuk setiap huruf agar visualisasi lebih mudah dibedakan. Papan kemudian digambar pada area kosong dengan setiap sel ditampilkan sebagai persegi berwarna sesuai warna nya.

Saat tombol Solve ditekan, GUI menginisialisasi proses brute force dan menjalankan fungsi solve() secara bertahap menggunakan mekanisme root.after(). Pendekatan ini memungkinkan proses pencarian solusi divisualisasikan secara live tanpa membuat GUI menjadi freeze.

Apabila solusi ditemukan, pengguna dapat memilih untuk save to text atau save to image.

2. Image Output

Kode:

```
def saveimg(self):
        if self.answer is None:
            messagebox.showerror("Error", "Belum ada solusi!")
            return

        dir =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
        savefolder = os.path.join(dir, "test")
        basename = self.filename.replace(".txt", "")
        savepath = os.path.join(savefolder,
f"{basename}_solution.png")
        size = 80
        imgsize = self.n*size
        image = Image.new("RGB", (imgsize, imgsize), "white")
        draw = ImageDraw.Draw(image)
```

```
        try:
            font = ImageFont.truetype("arial.ttf",
int(size/2))
        except:
            font = ImageFont.load_default()

        for r in range(self.n):
            for c in range(self.n):
                x1 = c*size
                y1 = r*size
                x2 = x1+size
                y2 = y1+size

                area = self.board[r][c]
                color = self.colors.get(area, "#ffffff")

                draw.rectangle([x1, y1, x2, y2], fill=color,
outline="black")

                if self.answer[r] == c:
                    text = "#"
                else:
                    text = area

                bbox = draw.textbbox((0, 0), text, font=font)
                w= bbox[2]-bbox[0]
                h = bbox[3]-bbox[1]

                draw.text(
                    (x1+(size-w)/2, y1+(size-h)/2),
                    text,
                    fill="black",
                    font=font
                )

        image.save(savepath)
        messagebox.showinfo("Saved", f"Saved to:\n{savepath}")
```

Penjelasan:

Fungsi saveimg() bertujuan untuk menghasilkan file gambar berformat PNG yang merepresentasikan solusi papan yang ditemukan. Gambar dibuat menggunakan library Pillow (PIL), khususnya objek Image dan ImageDraw.

Ketika fungsi dipanggil, program terlebih dahulu memeriksa apakah solusi

telah ditemukan. Jika belum, proses dihentikan. Selanjutnya, ukuran gambar dihitung berdasarkan ukuran papan (n × n), dengan setiap sel memiliki ukuran tetap (misalnya 80 piksel). Sebuah objek Image baru dibuat dengan mode RGB dan latar belakang putih. Program kemudian melakukan iterasi pada setiap sel papan. Untuk setiap sel, digambar sebuah persegi berwarna sesuai warna menggunakan metode draw.rectangle(). Jika posisi sel tersebut sesuai dengan posisi queen dalam solusi, maka simbol "#" digambar di tengah sel. Jika bukan, huruf warna asli tetap ditampilkan.

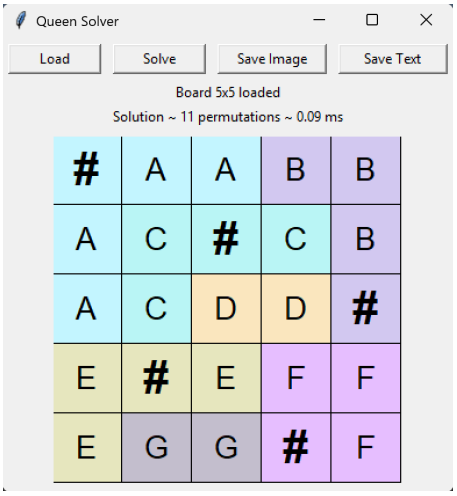
Setelah seluruh sel selesai digambar, gambar disimpan dalam format PNG ke folder tujuan dengan nama file yang menyesuaikan nama input awal.

Berikut adalah contoh solusi yang di save to image:

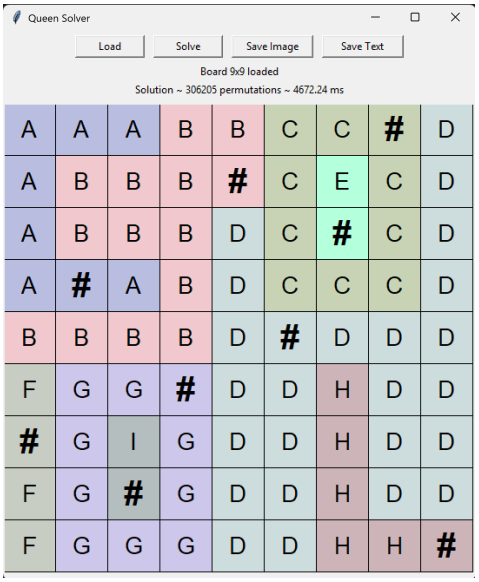
A	A	#	B	B	C	C	C	B
D	A	B	B	B	B	#	B	B
#	A	E	E	E	B	C	B	B
F	F	F	E	B	#	B	B	B
G	F	G	#	G	G	H	H	H
G	#	G	G	G	G	G	H	G
G	G	G	I	I	I	G	#	G
G	G	G	G	#	G	G	G	G
G	G	G	G	I	G	G	G	#

Pengujian dengan Test Case

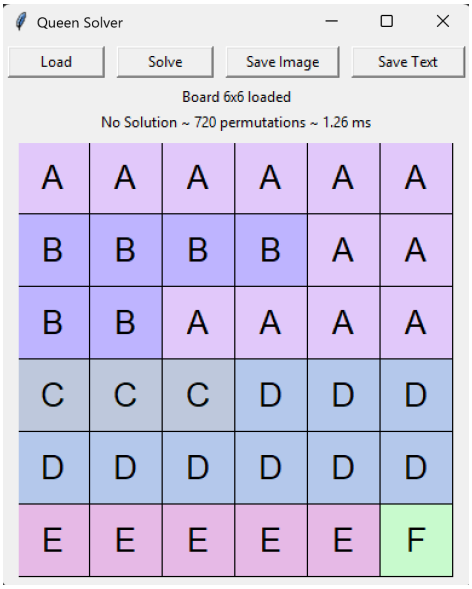
1. **Test Case #1** (5x5 Valid)

Input	Output
AAABB ACCCB ACDDB EEEFF EGGFF	

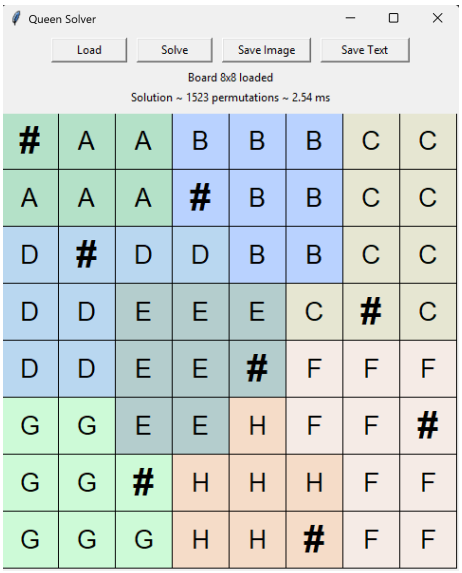
2. **Test Case #2** (9x9 Ada Solusi)

Input	Output
AAABBCCCD ABBBBCECD ABBBDCECD AAABDCCCD BBBBDDDDD FGGGDDHDD FGIGDDHDD FGIGDDHDD FGGGDDHHH	

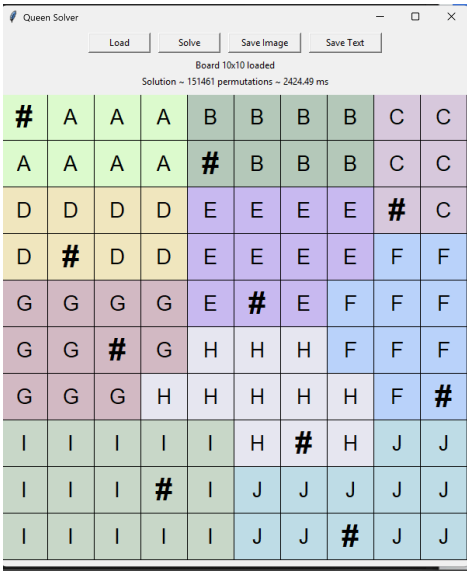
3. **Test Case #3** (6x6 Tidak Ada Solusi)

Input	Output
AAAAAA BBBBAA BBAAAA CCCDDD DDDDDD EEEEEF	

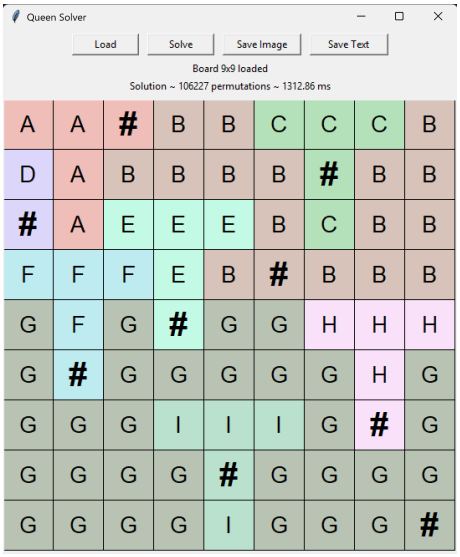
4. **Test Case #4** (8x8 Ada Solusi)

Input	Output
AAABBBCC AAABBBCC DDDDBBCC DDEEECCC DDEEEFFF GGEEHFFF GGGHHHFF GGGHHHFF	

5. **Test Case #5** (10x10 Ada Solusi)

Input	Output
AAAABBBBCC AAAABBBBCC DDDDEEEECC DDDDEEEEFF GGGGEEEFFF GGGGHHHFFF GGGHHHHHFF IIIIIHHHJJ IIIIIJJJJJ IIIIIJJJJJ	

6. **Test Case #6** (9x9 Ada Solusi)

Input	Output
AAABBCCCB DABBBBCBB DAEEEBCBB FFFEBBBBB GFGEGGHHH GFGGGGGHG GGGIIIGHG GGGGIGGGG GGGGIGGGG	

Lampiran

Tabel Pengerjaan:

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✔	
2	Program berhasil di jalankan	✔	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✔	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✔	
5	Program memiliki Graphical User Interface (GUI)	✔	
6	Program dapat menyimpan solusi dalam bentuk file gambar	✔	

Tautan Repositori: https://github.com/azzamrbni/Tucil1_18223025

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.

Muhammad Azzam Robbani